ShREEK Configuration User Guide
ShREEKInterface

Dave Evans
evansde@fnal.gov

27th September 2004

Contents

1 Overview

This document provides an overview of how to configure ShREEK via the `ShREEKInterface` Object. ShREEKInterface is a class that provides a python API for creating a ShREEK Configuration that can be saved and passed to the ShREEK Executor to control the execution of a set of tasks. Each aspect of the configuration is discussed, along with usage examples of the appropriate methods. Examples are provided for executing a single task and a list of tasks.

2 ShREEKInterface

ShREEK.ShREEKInterface

The object that provides the interface to ShREEK is the ShREEKInterface object and is defined in the module ShREEK.ShREEKInterface. The ShREEKInterface can be imported and instantiated, and the instance is then manipulated to create a configuration. A ShREEK Configuration is made up of a collection of python objects, but this is all hidden behind the ShREEK-Interface class.

To start creating a ShREEK Configuration one needs to import the ShREEKInterface object and create an instance of it. This can be done either within some python script as part of an API or by hand using the python interpreter.

An example illustrating the import and instantiation of a ShREEKInterface:

```
#  Import the ShREEKInterface object

from ShREEK.ShREEKInterface import ShREEKInterface

#  Now create an instance of ShREEKInterface
#  No arguments are required

myConfig = ShREEKInterface()

#  myConfig is now an empty ShREEK Configuration
#  that can be manipulated,
```

The ShREEKInterface constructor takes no arguments, and creates an empty ShREEK Configuration object that can then be filled with configuration information.

3 Adding Plugins

Since there are many potential features of an executable that need to be handled, monitored or controlled, ShREEK provides a Plugin mechanism that allows custom functionality to be loaded in and run during execution. These Plugins are python modules which are loaded to provide a set of objects that can be used to assist execution. The ShREEK Configuration contains the list of Plugin modules to be loaded for a given job, and this list of plugins can be manipulated via ShREEKInterface.

Two methods are provided to handle plugin manipulation.

- `ShREEK.ShREEKInterface.ShREEKInterface.listShREEKPlugins`
- `ShREEK.ShREEKInterface.ShREEKInterface.addPluginModule`

3.1 listShREEKPlugins

The listShREEKPlugins method allows you to list the Plugin modules in any particular instance of a ShREEKInterface object. The method takes no arguments and returns a python list containing the Plugin module names.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()

#  list the plugin modules in this object

plugins = myConfig.listShREEKPlugins()

#  list the plugin modules in this object

plugins = myConfig.listShREEKPlugins()

#  plugins is now a list of plugin module names
#  ShREEK contains some general plugin modules
#  which can be seen by looking at
#  an empty ShREEKInterface
for item in plugins:
    print item

#  sample output:
#   ShREEK.ShREEK_common.ProcStatUpdate
#   ShREEK.ShREEK_common.ShLoggerMonitor
#   ShREEK.ShREEK_common.PsUpdates
#   ShREEK.ShREEK_common.ReadProcStatus
#   ShREEK.ShREEK_common.StdoutMonitor
```

Note: The list of plugins returned by this method is a reference to the actual list which will be inserted into the configuration, so manipulation of this list is not recommended unless you are sure what you are doing.

3.2 addPluginModule

The addPluginModule method allows you to add the name of other plugin modules to the configuration file. The method takes a single argument, that is the name of the module as a string. This modulename must be available on the PYTHONPATH at runtime so that ShREEK can load it. Duplicate modules are filtered out silently.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()

#  Now imagine we want to add two plugin modules
#  The First named SomePlugin1.py,
#  The second named SomePackage/SomePlugin2.py,
#  where SomePackage is a python package directory
#  containing an __init__.py module and the plugin

#  add the first plugin
myConfig.addPluginModule(“SomePlugin1”)

#  and the second plugin:
myConfig.addPluginModule(“SomePackage.SomePlugin2”)

#  can verify the plugins exist with listShREEKPlugins
plugins = myConfig.listShREEKPlugins()

#  plugins will contain “SomePlugin1”
#  and “SomePackage.SomePlugin2”
```

4 Adding Tasks

In ShREEK, a Job is made up of a set of executable tasks. These tasks are managed as a series of lists. ShREEK Tasklists are built and managed via the ShREEKInterface. The ShREEKInterface object provides several methods for manipulation of the task structure for a particular job.

- `ShREEK.ShREEKInterface.ShREEKInterface.newTaskList`
- `ShREEK.ShREEKInterface.ShREEKInterface.addTask`

- `ShREEK.ShREEKInterface.ShREEKInterface.getTaskList`

The first tasklist that is added to the Interface instance will be the only one that is initially executed by the Executor in its default configuration, other tasklists are ignored unless installed by a ControlPoint plugin.

4.1 newTaskList

Before you can add any tasks to a ShREEKInterface configuration, you need to define a tasklist. A tasklist is essentially a named list that contains one or more executable tasks. To create a tasklist you use the newTaskList method to create a new task list with a unique name.

If you provide a name that already exists, then a ShREEKException will be raised.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()

#  Now create a new task list named "First"

myConfig.newTaskList("First")

#  Now create a new task list named "Second"

myConfig.newTaskList("Second")

#  Adding a duplicate tasklist named Second will
#  result in a ShREEKException
try:
    myConfig.newTaskList("Second")
except ShREEKException, ex:
    print ex
```

4.2 addTask

Executable tasks can be added to a task list using the addTask method. Tasks are kept in the order they are added, which is also the order of execution.

The default behaviour for addTask is to add tasks to the first tasklist unless the name of the tasklist is specified. The first argument is the name of the task script to be added, and the second optional argument is the name of the task list. Adding a task when no task lists have been specified leads to a ShREEKException.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()

#  Now create tasklists
myConfig.newTaskList(“First”)
myConfig.newTaskList(“Second”)

#  Now add a task to the First tasklist
myConfig.addTask(“./firstTask.sh”)
#  This is equivalent to doing
#  myConfig.addTask(“./firstTask.sh”,“First”)

#  Add another task to the First tasklist
myConfig.addTask(“./secondTask.sh”)

#  Add a task to the Second tasklist
myConfig.addTask(“./otherTask.sh”, “Second”)
```

4.3 getTaskList

The getTaskList method allows you to retrieve a reference to a task list by its name. This allows testing for the existence of a named tasklist, since if the name argument is not the name of an existing tasklist, then a ShREEKException will be raised.

This method is intended for use in reviewing the tasks that have been present, although power users may want to manipulate a task list directly.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()

#  Now create a tasklist
myConfig.newTaskList("First")

#  Get a reference to tasklist named First

taskList = myConfig.getTaskList("First")
for item in taskList:
    print item

#  If taskname is not present
#  a ShREEK Exception will be raised

try:
    taskList = myConfig.getTaskList("BadName")
except ShREEKException, ex:
    print ex
```

5 Configuring Monitoring

Monitors and Updators can be added and configured via the ShREEKInterface object. Updators are simply added according to the name that they are registered under. Monitors are added by registered name, but can also be configured with arguments and keyword options. Since the names of the monitors and updators are defined by the registration method called from the plugin module, you should check the documentation for plugins themselves to be sure you are getting the correct name.

- `ShREEK.ShREEKInterface.ShREEKInterface.addUpdator`
- `ShREEK.ShREEKInterface.ShREEKInterface.newMonitor`

- ShREEK.ShREEKInterface.ShREEKInterface.configureMonitor
- ShREEK.ShREEKInterface.ShREEKInterface.setMonitorParam
- ShREEK.ShREEKInterface.ShREEKInterface.setMonitorOption

5.1 addUpdator

The only interface that deals with Updators is the addUpdator method. The addUpdator Method takes a single argument, which is the name of the updator to be used. You must make sure that the updator name is registered from one of the plugin modules you provide.

The following example activates updator functions provided by the ShREEK_common subpackage. This Subpackage provides several plugins that are automatically registered.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()

#  Activate the Time updator
#  provided by the ShREEK_common tools
myConfig.addUpdator("Time")

#  Activate the PercentCPU updator
#  provided by the ShREEK_common tools
myConfig.addUpdator("PercentCPU")
```

5.2 newMonitor

The newMonitor Method creates a monitor configuration for a particular monitor of a certain type, identified by a unique name. The method takes two arguments, firstly, the unique name of the monitor, secondly, the type of the monitor.

The Name can be any string you choose, duplicate names will result in a ShREEKException. This name is used to configure the monitor. It is required to be unique so that multiple instances of the same monitor can be used, but with different configurations.

The type must be a registered monitor plugin name, which will be defined in the Plugin modules you load.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()

#  Create a stdout monitor named myStdout
#  the stdout monitor is defined in ShREEK_common

myConfig.newMonitor(“myStdout”, “stdout”)

#  Create a second stdout monitor named anotherStdout

myConfig.newMonitor(“anotherStdout”, “stdout”)
```

5.3 configureMonitor

Monitors are configured using a list of parameter args and a dictionary of option/value args. The actual monitor documentation will provide details on what options and parameters can be used. A monitor can be configured with a one-shot call to this method providing a set of parameters and options, or can be configured one step at a time using setMonitorParam and setMonitorOption.

The configureMonitor method requires that the first argument to it be the name of the monitor to be configured. If this name has not been defined with a newMonitor call, a ShREEKException will be raised.

The remaining arguments use python's built in handling of positional and optional arguments[1], via the *params and **options mechanism. The "second" argument can therefore be a set of positional parameters, to be passed as positional arguments, the "third" argument is a set of keyword = value pairs.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()

#  Create a stdout monitor named myStdout
myConfig.newMonitor(“myStdout”, “stdout”)

#  Configure the monitor with some positional args

myConfig.configureMonitor(“myStdout”, “Param1”, “Param2”)

#  Configure the monitor with some keyword args

myConfig.configureMonitor(“myStdout”,
                          Option1 = “Value1”,
                          Option2 = “Value2”)
```

5.4 setMonitorParam

The setMonitorParam method adds positional arguments to the configuration of a particular monitor. The first argument is the monitor name, then as many positional arguments can be specified as required.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()
myConfig.newMonitor(“myStdout”, “stdout”)

myConfig.setMonitorParam(“Param1”)
myConfig.setMonitorParam(“Param2”, “Param3”)
```

5.5 setMonitorOption

The setMonitorOption method allows a set of key-value options to be set for a particular monitor. The first argument is the monitor name, then as many keyword = value arguments can be specified as required.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()
myConfig.newMonitor(“myStdout”, “stdout”)

myConfig.setMonitorOption(Option1 = “Value1”)
myConfig.setMonitorOption(Option2 = “Value2”
                          Option3 = “Value3”)
```

6 Configuring Control Points

Documentation on Control Points will be included here when the implementation has been finalised.

7 Saving the Configuration

To be able to use the configuration object you have created you must save it to a file that can be passed to the ShREEK Executor. ShREEKInterface contains a method to do this:

- `ShREEK.ShREEKInterface.ShREEKInterface.writeShREEKConfig`

The writeShREEKConfig method takes a filename (with path if required) as its argument and writes out a ShREEK Configuration file reflecting the current state of the ShREEKInterface object to that location.

```
#  Create a ShREEKInterface object
from ShREEK.ShREEKInterface import ShREEKInterface
myConfig = ShREEKInterface()
#  ...
#  Configure the object as needed
#  ...

outputConfigFile = “./myShREEKConfig.xml”
myConfig.writeShREEKConfig(outputConfigFile)

#  The file myShREEKConfig.xml now exists in your
#  current working directory and can be used to
#  configure and run ShREEK
```

8 Executing the Configuration

To Execute a configuration file with the ShREEK Executor, you need to have the configuration file written out (see §7) and then invoke the executor with the file as follows:

- python Executor.py –taskcfg=<path to Configuration File>

9 Example: Running a single task

10 Example: Running a list of tasks

TBA

References

[1] Python Calls mechanism.
`http://www.python.org/doc/current/ref/calls.html#calls`